Laboratorul 03.

Problema 1

Definiți o clasă ***Fractie*** care modelează lucrul cu fracțiile. Membrii acestei clase sunt:

- două atribuite de tip ***int*** pentru numărătorul fracției, respectiv numitorul ei;
- constructorul cu doi parametri de tip ***int***, pentru setarea celor două parți ale fracției (numărător și numitor);
- un constructor fără parametri care apelează constructorul anterior;
- o metodă, cu un **singur** parametru, de calcul a sumei a două fracții;
- o metodă ***toString*** uzitată pentru afișarea pe ecran a valorii fracției;
- o metodă ***equals***, cu un parametru de tip ***Object***, care va returna ***true*** dacă fracțiile sunt egale, respectiv ***false*** în sens contrar;
- o metodă ***main*** pentru testarea funcționalității clasei.

Consultați exemplul prezentat la curs pentru numere complexe!

Problema 2

Un produs este caracterizat prin nume (***String***), preț (***double***) și cantitate (***int***). Un magazin are un nume (***String***) și conține 3 produse. Creați clasele ***Produs*** și ***Magazin*** corespunzătoare specificațiilor de mai sus. În fiecare clasă, implementați constructorul potrivit, astfel încât caracteristicile instanțelor să fie setate la crearea acestora. Clasa ***Produs*** conține o metodă ***toString***, care va returna un ***String*** sub forma **"Produs <nume_produs> <preț_produs> <cantitate>"** și o metodă ***getTotalProdus*** care va returna un ***double***, produsul dintre cantitate și preț. Clasa ***Magazin*** conține o metodă ***toString*** care va returna String-ul corespondent tuturor componentelor magazinului și o metodă ***getTotalMagazin*** care va calcula suma totalurilor produselor și o va returna. Creați, într-o metodă ***main***, un obiect de tip ***Magazin***, uzitând obiecte anonime în cadrul instanțierii.

```
Patrat p = new Patrat(new Point(0, 0), new Dimension(5, 5));
```

Problema 3

Să se definească o clasă ***MyQueue*** care să descrie o structură de date de tip coadă. Datele clasei (private):

- un obiect de tip ***MyArray*** (clasa inclusă în arhiva laboratorului);
- o constantă (***Infinit***) având valoarea 9500;
- indicele primului element din coadă;
- indicele ultimului element din coadă;
- numărul de elemente din coada.

Constructorul clasei:

- constructor fără parametri care se ocupă de inițializările membrilor.

Metodele clasei:

- ***int getSize()*** = are ca rezultat numărul de elemente din coadă;
- ***void enqueue(int value)*** = adaugă o valoare în coadă;

- ***int dequeue()*** = întoarce primul element din coadă și îl elimină, incrementând indicele corespunzător, fără a elimina efectiv elementul din obiectul de tip ***MyArray*** (***Infinit*** - coada vidă);
- ***boolean isEmpty()*** = verifică dacă este vidă coada;
- ***String toString()*** = String cu elementele din structura de date.

Pentru verificare, se va folosi clasa de mai jos.

```
class MyArray {
    private int v[];
    private int size;

    public MyArray() {
        this(100);
    }

    public MyArray(int length) {
        size = 0;
        v = new int[length];
    }

    public int get(int poz) {
        if(poz < size) {
            return v[poz];
        } else {
            return -1;
        }
    }

    public void set(int pos, int value) {
        v[pos] = value;
        size++;
    }

    public int getSize() {
        return size;
    }
}


class Test {
    public static void main(String args[]) {
        MyQueue queue = new MyQueue();
        queue.enqueue(7);
        queue.enqueue(8);
        queue.enqueue(10);
        queue.enqueue(-1);
        queue.enqueue(2);
        System.out.println(queue);
        System.out.println(queue.dequeue());
        System.out.println(queue.getSize());
        System.out.println(queue);
        queue.enqueue(9);
        queue.enqueue(queue.dequeue());
        queue.enqueue(11);
        queue.enqueue(22);
        System.out.println(queue);
        while(!queue.isEmpty()) {
            System.out.print(queue.dequeue() + " ");
        }
        System.out.println("");
        System.out.println(queue);
    }
}
```

Problema 4

Definiți o clasă ***Numar*** care are ca membru un număr întreg și conține metodele descrise mai jos. Implementați metodele astfel încât fiecare metodă să efectueze o singură adunare. Instanțiați un obiect de

tip ***Numar*** în metoda ***main*** și apelați metodele implementate. Ce principiu POO este evidențiat în acest exercițiu?

```
//returneaza suma dintre nr (membrul clasei) si a
public int suma(int a);
//returneaza suma dintre nr, a si b
public int suma(int a, int b);
//returneaza suma dintre nr, a, b si c
public int suma(int a, int b, int c);
//returneaza suma dintre nr, a, b, c si d
public int suma(int a, int b, int c, int d);
```

Problema 5

Implementați clasa ***Punct*** care definește un punct din spațiul 2D.

Datele clasei (private):

- două nr. întregi reprezentând cele două coordonate ale punctului.

Conctructorul clasei:

- un constructor fără parametri care instanțiază punctul `O(0, 0)`.

Metodele clasei:

- `int getX()` = întoarce abscisa punctului;
- `void setX(int x)` = seteaza abscisa punctului;
- `int getY()` = întoarce ordonata punctului;
- `void setY(int y)` = setează ordonata punctului;
- `String toString()` = returnează un String de forma `(x, y)`;
- `double distance(int, int)` = calculează distanța dintre 2 puncte;
- `double distance(Punct p1)` = calculează distanța dintre 2 puncte.

Creați o clasă ***Test***, în același pachet cu clasa ***Punct***, care conține o metodă ***main*** care calculează distanța dintre punctele `A(1, 2)` si `B(-1, 3)`.

Puteți accesa datele clasei ***Punct*** în metoda ***main*** din clasa ***Test***?

Problema 6

Să se definească o clasă ***Graph*** care să descrie un graf ponderat orientat care are nodurile numerotate de la 1.

Datele clasei (private):

- o matrice cu componente de tip int (matricea costurilor) - matrice;
- o constantă (***Infinit***) având valoarea 9500;
- numărul de noduri - n .

Constructorul clasei:

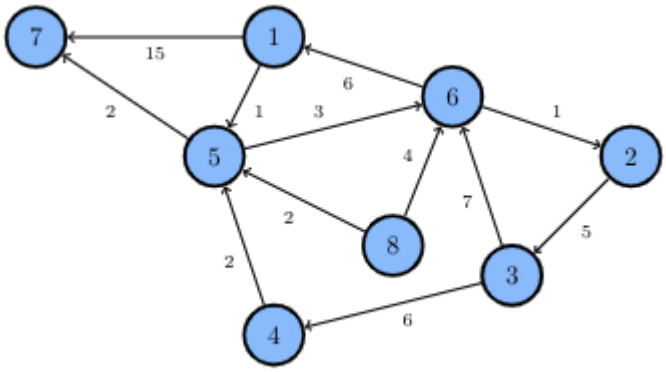
- constructor cu un parametru întreg (numărul de noduri din graf)

Metodele clasei:

- `int getSize()` = are ca rezultat numărul de noduri din graf;

- `void addArc(int v, int w, int cost)` = adaugă un arc la graf (între v și w, având costul $cost$);
- `boolean isArc(int v, int w)` = verifică dacă există arc între v și w în graf;
- `toString()` = afișarea grafului (se va alege o variantă intuitivă de afișare a grafului);
- `int[][] floydWarshall()` = implementarea algoritmului \textit{Floyd - Warshall} pentru determinarea drumurilor de cost minim în graf;
- `void main(String[])` = metoda main pentru testarea functionalității clasei implementate.

Pentru verificare, se va crea graful din figura de mai jos. Se va afișa graful, uzitând metoda ***toString*** implementată, și se va determina distanța minimă dintre două noduri folosind algoritmul **Floyd - Warshall**. Folosiți scheletul de mai jos:

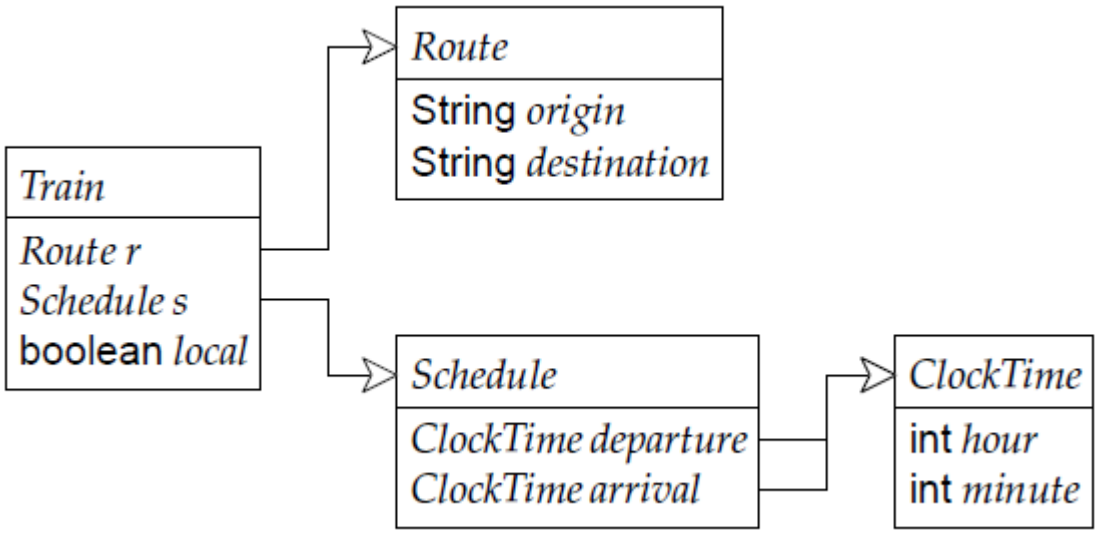



```
class Graph {
    //....

    public int[][] floydWarshall() {
                int result[][];
                result = new int[n+1][n+1];
                int k, i, j;
                for(i = 1; i <= n; i++) {
                        for(j = 1; j <= n; j++) {
                                if(i == j) {
                                        result[i][j] = 0;
                                } else if(isArc(i, j)) {
                                        result[i][j] = matrice[i][j];
                                } else {
                                        result[i][j] = Infinit;
                                }
                        }
                }
                for(k = 1; k <= n; k++) {
                        for(i = 1; i <= n; i++) {
                                for(j = 1; j <= n; j++) {
                                        int dist;
                                        dist = result[i][k] + result[k][j];
                                        if(result[i][j] > dist) {
                                                result[i][j] = dist;
                                        }
                                }
                        }
                }
                return result;
        }

        public static void main(String args[]) {
                Graph g = new Graph(4);
                g.addArc(1, 3, 2);
                g.addArc(1, 2, 3);
                g.addArc(2, 4, 6);
                g.addArc(2, 3, 2);
                System.out.println(g);
                System.out.println("Floyd-Warshall");
```

```
                int [][] my_matrix = g.floydWarshall();
                System.out.println("distanta minima dintre nodurile 1 si 4 este "+ my_matrix[1][4]); // rezultat - 9
        }
}
```

Problema 7

Să se implementeze ierarhia de clase descrisă prin următoarea diagramă:



Clasa ***Schedule*** conține o metodă care calculează durata călătoriei în minute. Știind algoritmul de calcul al prețului unui bilet de călătorie, să se implementeze o metodă, în clasa ***Train***, care calculează prețul unui bilet. Valoarea unui bilet este egală cu ***X * durata_călătoriei***, unde X este egal cu 1 pentru cursele interne și 2 pentru cursele internaționale. Clasa ***Route*** va conține un constructor cu 2 parametri și o metodă care primește ca parametru un obiect de tip ***Route*** și verifică dacă sunt de tip tur - retur cele două rute, rezultatul fiind de tip ***boolean***. Clasa ***ClockTime*** conține o metodă, cu un parametru de tip ***ClockTime***, și compară două momente de timp, rezultatul fiind un ***int***.

Adaugați o metodă ***main*** pentru testarea claselor implementate, utilizând exemplele oferite. Definiți un constructor potrivit pentru instanțierea unui obiect de tip ***Train***, în care să apelați constructorii definiți în clasele ***Route***, ***Schedule*** și ***ClockTime***.

```
[local] [origin (departure)] -> [destination (arrival)]
true Bucuresti Nord (9:35) -> Constanta (12:02)
true Bucuresti Nord (5:45) -> Iasi (12:49)
false Bucuresti Nord (23:45) -> Sofia (17:00)
```